Theor Chem Acc (2007) 117: 73–84
DOI 10.1007/s00214-006-0145-x

REGULAR ARTICLE

George S. Fanourgakis · Vinod Tipparaju
Jarek Nieplocha · Sotiris S. Xantheas

An efficient parallelization scheme for molecular dynamics simulations with many-body, flexible, polarizable empirical potentials: application to water

Received: 14 March 2006 / Accepted: 31 March 2006 / Published online: 13 July 2006

Abstract An efficient parallelization scheme for classical molecular dynamics simulations with flexible, polarizable empirical potentials is presented. It is based on the standard Ewald summation technique to handle the long-range electrostatic and induction interactions. The algorithm for this parallelization scheme is designed for systems containing several thousands of polarizable sites in the simulation box. Its performance is evaluated during molecular dynamics simulations under periodic boundary conditions with unit cell sizes ranging from 128 to 512 molecules employing two flexible, polarizable water models [DC(F) and TTM2.1-F] containing 1 and 3 polarizable sites, respectively. The time-to-solution for these two polarizable models is compared with the one for a flexible, pairwise-additive water model (TIP4F). The benchmarks were performed on both shared and distributed memory platforms. As a result of the efficient calculation of the induced dipole moments, a superlinear scaling as a function of the number of the processors is observed. To the best of our knowledge, this is the first reported results of parallel scaling and performance for simulations of liquid water with a polarizable potential under periodic boundary conditions.

Keywords Parallel computing · Molecular dynamics · Empirical potentials · Polarizable models · Water

G. S. Fanourgakis · S. S. Xantheas (✉)
Chemical Sciences Division,
Pacific Northwest National Laboratory,
902 Battelle Boulevard, PO Box 999, MS K1-83,
Richland, WA 99352, USA
E-mail: sotiris.xantheas@pnl.gov

V. Tipparaju · J. Nieplocha
Computational Science and Mathematics,
Pacific Northwest National Laboratory,
PO BOX 999, MS K1-85,
Richland, WA 99352, USA

1 Introduction

Advances in computer technology over the last decade made it possible to render a molecular level picture into the study of systems of chemical and biological interest using molecular dynamics (MD) and monte carlo (MC) techniques. Current Massively Parallel Processing (MPP) architectures and computer clusters have enabled new progress in these areas.

The efficient parallelization of simulation algorithms can provide a powerful tool for assessing the accuracy of various simulation techniques. This is because statistical uncertainties during a simulation decrease as both the size of the system and the simulation length increase, while several approximations that are necessary due to the lack of computer power are avoided. Once these issues are sorted out it becomes possible to evaluate the accuracy of reduced representations of the intra- and intermolecular interactions, such as classical interaction potentials, which are based on different physics.

Several toolkits have been developed over the last few years in order to leverage the power of parallel computer systems and provide the programmer with high level tools for the fast development of parallel codes. For instance, the message passing interface (MPI) [1] and global arrays (GA) toolkits [2–4] have been extensively used for the development of several MD and first principles (ab-initio) suites of codes.

Molecular dynamics simulations can be computationally demanding, depending on the size of the system (number of atoms) and/or the length of simulation (number of time steps). Biological molecules usually consist of a large number of atoms, while for a realistic description of their dynamics an aqueous environment is required. The size of the system thus typically consists of hundreds of thousands of atoms. For biological systems of that size, parallel implementations of polarizable force fields already exist in several MD packages, such as CHARMM [5] and AMBER [6].

On the other hand, some macroscopic properties of several systems of interest to the chemical physics community can be successfully simulated using supercells of only a few hundred atoms or molecules or even less. For this size regime, as the computer memory requirements are significantly smaller,

it is possible to design alternative parallelization schemes and therefore decrease even more the time-to-solution. For example, it has been found that unit cells of just 32 [7] or 54 [7,8] molecules can produce accurate estimates for properties such as the enthalpy and the radial distribution function of liquid water under periodic boundary conditions (PBC). Typical simulations of liquid water under PBC are based upon simulation cells of 200–256 water molecules while for some properties no more than 512 molecules are required. For these cases, however, the length of the simulation is much more crucial than the system size. For example, flexible water empirical potential models, which allow for the explicit description of the intramolecular stretching and bending motions, require typical time steps of $\delta t = 0.2$ fs or less. Furthermore, properties like the dielectric constant require at least $1-2$ ns of simulation time or about 10^7 time steps. These conditions can currently be met using simple pairwise–additive interaction potentials, however for the more computationally demanding many-body polarizable models this task can amount to more than several months of real time on a single processor. The time-to-solution is clearly an issue for system sizes of a few thousand atoms such as the ones considered in this study.

Water is by far the most studied liquid due to its importance in several different fields. The first computer simulations of liquid water using simple pairwise-additive empirical potentials were performed about three decades ago [9,10]. The electrostatic interactions between water molecules are usually modeled by assigning fractional charges to the oxygen and hydrogen atoms, while the electronic repulsion and the dispersion interactions are typically described by pair–additive (usually Lennard–Jones) interactions between the Oxygen atoms.

This work is focused on the efficient parallelization of many-body, flexible, polarizable interaction potentials for water for use in MD simulations of the liquid properties under PBC. Our parallelization strategy works best with distributed memory systems and also shared non-uniform memory (NUMA) systems. To the best of our knowledge there is no prior published results regarding the scaling of algorithms used for the parallelization of polarizable potentials for water under PBC. The material is organized as follows: In Sect. 2 expressions of the energy and forces are presented for a periodic system with charges and polarizable atoms in which the Ewald summation technique is employed. In Sect. 3 we discuss several basic schemes, which are appropriate for the parallelization of the simpler pairwise–additive potentials. Based on the ideas presented in that section, we furthermore present in Sect. 4 an efficient scheme for the parallelization of a polarizable model under PBC. In Sect. 5 we describe three different water models of increasing complexity that were used as test cases in order to evaluate the efficiency of the proposed algorithm, as well as the computer platforms that the benchmarks were performed and the results of the benchmark calculations. These results are discussed in Sect. 6 where conclusions and guidelines for future development are also presented.

2 Ewald summation for charges and polarizable atomic dipoles

The Ewald summation has been used to treat the long-range interactions between charges and induced dipoles. Details of the Ewald method have been presented previously by Nymand and Linse [11]. Here we outline the basic equations in a form that is suitable to discuss the basic concepts of our parallel scheme in the subsequent sections.

The total energy of a system that contains charges and dipoles is given by the sum of the electrostatic and induction energies:

$$\begin{aligned} U &= U_{\text{elec}} + U_{\text{ind}} \\ &= \left\{ \frac{1}{2} \sum_i q_i \phi_i^{\text{stat}} \right\} + \left\{ -\frac{1}{2} \sum_i \mu_{i,\alpha} E_{i,\alpha}^{\text{stat}} \right\}. \end{aligned} \tag{1}$$

In this equation q_i is the charge and $\mu_{i,\alpha}$ is the $\alpha (= x, y, z)$ component of the induced dipole moment of atom i (for simplicity we will consider that the system does not contain permanent dipoles). For the Greek letter indices the Einstein summation convention is followed. ϕ_i^{stat} and $E_{i,\alpha}^{\text{stat}}$ are the (static) electrostatic potential and the electrostatic field on atom i arising only from the charges of the system. The total electrostatic potential, ϕ_i, and field, $E_{i,\alpha}$, also include an induction term. The expression for the total electrostatic potential and field is given by:

$$\begin{aligned} \phi_i &= \phi_i^{\text{stat}} + \phi_i^{\text{ind}} \\ &= \left\{ \sum_{j \neq i} \hat{T}_{ij} q_j - \frac{2\kappa}{\sqrt{\pi}} q_i + \frac{4\pi}{V} \sum_{\mathbf{k} \neq \mathbf{0}} A_k \text{Re}(\text{e}^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^q) \right\} \\ &\quad + \left\{ \sum_{j \neq i} \hat{T}_{ij}^{\alpha} \mu_{j,\alpha} + \frac{4\pi}{V} \sum_{\mathbf{k} \neq \mathbf{0}} A_k \text{Re}(\text{e}^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^{\mu}) \right\}, \end{aligned} \tag{2}$$

$$\begin{aligned} E_{i,\alpha} &= E_{i,\alpha}^{\text{stat}} + E_{i,\alpha}^{\text{ind}} \\ &= \left\{ \sum_{j \neq i} \hat{T}_{ij}^{\alpha} q_j - \frac{4\pi}{V} \sum_{\mathbf{k} \neq \mathbf{0}} A_k k_\alpha \text{Re}(i\text{e}^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^q) \right\} \\ &\quad + \left\{ \sum_{j \neq i} \hat{T}_{ij}^{\alpha\beta} \mu_{j,\beta} + \frac{4\pi}{3V} \mu_{i,\alpha} \right. \\ &\quad \left. - \frac{4\pi}{V} \sum_{\mathbf{k} \neq \mathbf{0}} A_k k_\alpha [\text{Re}(i\text{e}^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^{\mu}) - \mu_{i,\beta} k_\beta] \right\}. \end{aligned} \tag{3}$$

For an atom with polarizability a_i the induced dipole moment, $\mu_{i,\alpha}$, is given by:

$$\mu_{i,\alpha} = \alpha_i E_{i,\alpha}^{\text{stat}} + \alpha_i E_{i,\alpha}^{\text{ind}}. \tag{4}$$

The tensors in Eqs. (2) and (3) are: $\hat{T}_{ij} = \text{erfc}(\kappa r_{ij})/r_{ij}$, $\hat{T}_{ij}^{\alpha} = \nabla_\alpha \hat{T}_{ij}$, $\hat{T}_{ij}^{\alpha\beta} = \nabla_\alpha \nabla_\beta \hat{T}_{ij}$, and $\hat{T}_{ij}^{\alpha\beta\gamma} = \nabla_\alpha \nabla_\beta \nabla_\gamma \hat{T}_{ij}$, where κ is the Ewald parameter, $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ where $\mathbf{r}_i$ is the

position vector of atom i, and $\mathrm{erfc}(x)$ is the complementary error function ($\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$, where $\mathrm{erf}(x)$ is the error function). It should be mentioned, however, that the $\hat{T}_{ij}$ in the previous form is appropriate only for atomic systems or intermolecular interactions. For the case of a molecule in which no electrostatic interaction between two atoms within the molecule is considered, the form $\hat{T}_{ij} = \mathrm{erf}(\kappa r_{ij})/r_{ij}$ should be used instead.

Both the static and induced terms for ϕ_i and $E_{i,\alpha}$ of Eqs. (2) and (3) include a sum in the reciprocal space which accounts for the long-range corrections. For a primary box of lengths L_x, L_y, and L_z and volume V, the k-vectors are given by:

$$\mathbf{k} = 2\pi \left(\frac{n_x}{L_x}, \frac{n_y}{L_y}, \frac{n_z}{L_z} \right). \tag{5}$$

The n_x, n_y, n_z are integers and their maximum values are determined by the size of the box and the required accuracy. In Eqs. (2) and (3) we have defined A_k as:

$$A_k = k^{-2} \mathrm{e}^{-k^2/4\kappa^2}, \tag{6}$$

while the complex Q^q and Q^μ ($\bar{Q}^q$ and $\bar{Q}^\mu$ are their corresponding complex conjugates) are given by:

$$\begin{aligned} Q^q &= \sum_j q_j \mathrm{e}^{i\mathbf{k}\cdot\mathbf{r}_j}, \\ Q^\mu &= \sum_j i(\boldsymbol{\mu}_j \cdot \mathbf{k}) \mathrm{e}^{i\mathbf{k}\cdot\mathbf{r}_j}, \end{aligned} \tag{7}$$

Finally the force on atom i is given by:

$$f_{i,\alpha} = q_i E_{i,\alpha} + \mu_{i,\beta} E_{i,\beta\alpha}, \tag{8}$$

where, $E_{i,\alpha\beta}$ is the electrostatic field gradient:

$$\begin{aligned} E_{i,\alpha\beta} &= E^{\mathrm{stat}}_{i,\alpha\beta} + E^{\mathrm{ind}}_{i,\alpha\beta} \\ &= \left\{ -\sum_{j\neq i} \hat{T}^{\alpha\beta}_{ij} q_j - \frac{4\pi}{3V} q_i \delta_{\alpha\beta} \right. \\ &\quad \left. + \frac{4\pi}{V} \sum_{\mathbf{k}\neq\mathbf{0}} A_k k_\alpha k_\beta [\mathrm{Re}(e^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^q) - q_i] \right\} \\ &\quad + \left\{ -\sum_{j\neq i} \hat{T}^{\alpha\beta\gamma}_{ij} \mu_{j,\gamma} + \frac{4\pi}{V} \sum_{\mathbf{k}\neq\mathbf{0}} A_k k_\alpha k_\beta [\mathrm{Re}(\mathrm{e}^{i\mathbf{k}\cdot\mathbf{r}_i} \bar{Q}^\mu)] \right\}. \end{aligned} \tag{9}$$

3 Parallelization of a pairwise–additive potential

There exist two main strategies for the parallelization of a pairwise–additive potential, namely *atom-decomposition* and *force-decomposition*. Several algorithms belonging to these two classes have been proposed and implemented in MD packages (for more details see Refs. [12–15] and the references therein). As it will become evident in the next section, it is important to determine which of those is appropriate to be used in the case of a polarizable potential. For this reason, in this section we shortly describe possible parallelization schemes for a pairwise–additive potential.

We first consider the simpler case of a periodic system with N atoms that interact via a short-range pairwise-additive potential. The total energy of the system and the force on atom i are given by:

$$\begin{aligned} E &= \frac{1}{2} \sum_i \sum_{j\neq i} \epsilon_{ij} = \sum_i \sum_{j>i} \epsilon_{ij}, \\ \mathbf{f}_i &= \sum_{j\neq i} \mathbf{f}_{ij}, \end{aligned} \tag{10}$$

where $\epsilon_{ij} = \epsilon(r_{ij})$ is the pair interaction energy between the atoms i and j, while $\mathbf{f}_{ij} = \mathbf{f}(\mathbf{r}_{ij})$ is the corresponding force. Once the force on each atom i is determined at time t, its position $\mathbf{r}_i$ and velocity $\mathbf{v}_i$ can be obtained for time $t + \delta t$ by integrating Newton's equations:

$$\begin{aligned} m_i \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} &= \mathbf{f}_i, \\ \frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} &= \mathbf{v}_i. \end{aligned} \tag{11}$$

We will assume that the interaction between two atoms is zero if the distance between them is larger than a certain cutoff distance, R_c, which does not exceed half the size of the box, L ($R_\mathrm{c} \leq L/2$).

Due to the fact that the time step δt is small, a specific atom usually interacts with the same set of atoms for several time steps. We can take advantage of this fact for the fast evaluation of the energy and forces using the neighbour list method proposed by Verlet [16]. In this method, for each atom a list containing labels of all atoms within a sphere of radius R_s centered on that atom is created. The value of R_s should be chosen to be larger than R_c. At each time step, instead of examining if all atoms of the system are inside the sphere of radius R_c, only the atoms within the sphere of radius R_s are examined. It is clear that depending on the conditions of the simulation (i.e. density and temperature) and the difference ($R_\mathrm{s} - R_\mathrm{c}$), after a certain number of time steps the list of neighbours for each atom should be updated. Details of the implementation of the neighbour list algorithm can been found in Ref. [17].

Clearly the most straightforward parallel implementation of the system (scheme (a) in Fig. 1) relies on assigning a group of N/P atoms to each one of the available P processors. Initially, each processor has to calculate the Verlet list for these atoms and compute the forces in order to update the positions $\mathbf{r}_i$ by integrating Newton's equations. Communication among all processors is required at this point in order to exchange the updated atom positions. The disadvantage of this method is that it does not take into account Newton's third law, namely $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$. As a result, the total number of computations is doubled. A second disadvantage is that this technique does not provide an opportunity to balance the amount of work done by each processor (load balancing). Each processor will have equal amount of work only

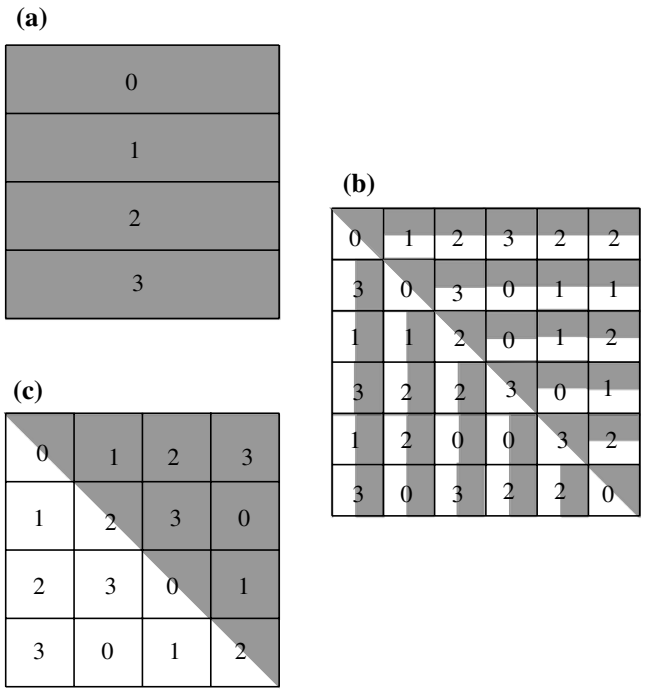

Fig. 1 Illustrative examples of the parallelization schemes (**a**), (**b**) and (**c**) described in the text. We have considered $P = 4$ processors labeled from 0 to 3. The total $N \times N$ pair interactions are divided in smaller blocks. Every processor performs calculations only in the shaded area of each block

if the number of the total interactions for the N/P atoms is the same. However, depending on the physical system that is under study and the conditions of the simulation, it is possible that the number of neighbours will differ significantly for each atom. As a result, the work load will be different for each processor. Lack of balanced per-processor load can have a significant and adverse impact on the overall simulation time [18].

An alternative parallelization scheme (Fig. 1b) that overcomes the difficulties described above requires a small amount of additional communication. In this scheme the total workload is divided in $(N/q) \times (N/q)$ indexed blocks. The efficiency of the calculation is related to the value of the parameter q, as will become clear in the subsequent discussion. An example is shown in Fig. 1b, for $q = 6$: in this case the total array is divided into 36 blocks, indexed from 0 to 35. In order to determine the index of the block a processor is to work on next, a globally shared counter variable is maintained [19]. Its current value represents the index of the next block that needs to be worked on. When a processor is finished with its current block, it increments the counter again and obtains the index of the next block to work on. This process is repeated until the counter reaches a value greater than the number of blocks, thus indicating that the computations on all blocks have been completed. Since the counter is globally shared and multiple processors can attempt to access it concurrently, the updates to the counter value must be "atomic" (i.e., executed in a serial manner) in order to ensure consistency of the counter values. In our implementation the "atomic" update is supported through the Global Arrays *read_and_increment* operation. As it is also shown in Fig. 1b the blocks as well as the number of blocks that each processor is going to work on are not predetermined. This depends on the number of interactions that needs to be computed in each block. A processor working on a block with more interactions than the other blocks may end up working with less blocks than the other processors. This is how a balance in the load for each processor is achieved. In the previous example the processors 0, 1, 2, 3 worked on 9, 8, 11 and 8 blocks, respectively. However, it is important to carefully choose the number of blocks that the whole array will be divided to as this determines the degree of load balancing that can be achieved. A small number of blocks will not guarantee load-balancing, while a large number of blocks may be better for load balance but may also increase the amount of communication slowing down the entire calculation. Using this method it is easy to take advantage of Newton's third law, reducing the amount of computations by half with respect to the previous method. A simple way to achieve this is shown in Fig. 1b, in which each processor performs computations only for the shaded areas. In this way for a pair of atoms i and j only the force $\mathbf{f}_{ij}$ is calculated (not $\mathbf{f}_{ji}$), while the total force on these atoms will be updated according to $\mathbf{f}_i = \mathbf{f}_i + \mathbf{f}_{ij}$ and $\mathbf{f}_j = \mathbf{f}_j - \mathbf{f}_{ij}$. After the end of all computations each processor contains only a part of the force for every atom. A global summation of the forces among the processors is required at this point in order to be able to update the positions of the atoms according the Eq. (11).

Finally a third scheme is illustrated in Fig. 1c. In this approach the total workload is divided in $(N/P) \times (N/P)$ blocks. All processors are working only on the whole block $B_{a,b}$ if $b > a$, while if $b = a$ the processor calculates only the upper diagonal part of interactions. In contrast to the previous scheme, each processor works on predetermined blocks. For the first row of blocks the processors 0 through $P - 1$ are assigned, while for the second row a cyclic left shift is performed with respect to the first row and so on. As it can be seen in the example of Fig. 1c, we can still take advantage of Newton's third law by calculating only specific interactions. According to the previous discussion the workload for each processor may be different. However, as we will see in the next Section, it is possible to overcome this problem.

4 Parallelization of a many-body polarizable potential

The procedure for the calculation of the energy and forces of a periodic system containing charges and dipoles can be divided in the following three steps:

1. Calculation of the ϕ_i^{stat}, $E_{i,\alpha}^{\text{stat}}$ and $E_{i,\alpha\beta}^{\text{stat}}$ from the first part of the Eqs. (2), (3) and (9) and construction of the dipole moment tensor array $\hat{T}_{ij}^{\alpha\beta}$,
2. Calculation of the total electric field $E_{i,\alpha}$ by solving the linear system of Eq. (3) and the induced dipole moment $\mu_{i,\alpha}$ given by Eq. (4),

3. Calculation of the induction ϕ_i^{ind} and $E_{i,\alpha\beta}^{\text{ind}}$ terms [second part of the Eqs. (2), (9)] and subsequently the total energy U, forces $f_{i,\alpha}$, [Eqs. (1), (8)] and virial.

We observe that the steps (1) and (3) are equivalent, in the sense that in the real space only interactions between pairs of atoms are considered, i.e. in step (1) the pairwise–additive charge–charge interactions are calculated, while in step (3) the pairwise–additive charge–dipole and dipole–dipole interactions are computed. For these calculations in real space, one of the parallel algorithms described in the previous section may be used. However, it is step (2) that is the most time consuming part of the whole procedure as the induced dipole moment $\mu_{i,\alpha}$ on each atom i is a function of all atoms of the system (many-body potential). In order to compute the $\mu_{i,\alpha}$, for a system that contains N_d polarizable atoms, a linear system of $(D \times D)$ equations should be solved ($D = 3N_d$ where the factor 3 comes from the three components x, y, z of each dipole). Equation (3) may be written as $\mathbf{AE} = \mathbf{E}^{\text{stat}}$, where $\mathbf{E}^{\text{stat}}$ is a vector of size D that contains all $E_{i,\alpha}^{\text{stat}}$ elements, and has been already calculated in step (1). By taking into account that the array $\mathbf{A}$ is symmetric, one may use the Cholesky method (see for example Reference [20]) in order to solve the system of equations. An alternative approach is to use an iterative procedure. Assuming that a good initial estimate, $\mathbf{E}^{(0)}$, of the total electric field $\mathbf{E}$ can be made, a new electric field, $\mathbf{E}^{(1)}$, closer to $\mathbf{E}$ can been computed by substituting $\mathbf{E}^{(0)}$ in the right-hand side of Eq. (3). This procedure should be repeated k times, until the desired accuracy of $\mathbf{E}^{(k)}$ is reached. As the electric field $\mathbf{E}^{\text{stat}}$ arising from the charges is usually much more significant than the field arising from the induced dipoles $\mathbf{E}^{\text{ind}}$, one can set $\mathbf{E}^{(0)} = \mathbf{E}^{\text{stat}}$. However, even better choices of $\mathbf{E}^{(0)}$ have been proposed and are discussed in the following sections. The Cholesky method requires $D^3/2$ operations while the iterative procedure $k\,D^2$. A detailed discussion of the two methods can be found in Ref. [21]. In our implementation, as the number of iterations k is much smaller than $D/2$, we have used the iterative procedure.

By examining the right-hand side of Eq. (3) it can be realized that the matrix elements of dipole tensor matrix $\hat{T}_{ij}^{\alpha\beta}$ can be calculated and stored in the memory before step (2), in this way avoiding the re-calculation of the same matrix elements during the iterative procedure. For this reason in our implementation we have included this calculation in step (1). The matrix–vector multiplication of the $\hat{T}_{ij}^{\alpha\beta}$ and $\mu_{j,\beta}$ will also be performed much faster as each processor will be able to use efficiently its cache memory. However, for the parallel execution of the matrix–vector multiplication, attention should be paid to the following points: (i) The matrix elements of the $\hat{T}_{ij}^{\alpha\beta}$ tensor should be distributed equally among the processors for load balance, (ii) any transfer of the matrix elements of $\hat{T}_{ij}^{\alpha\beta}$ between processors should be avoided and (iii) the symmetry of the array $\hat{T}_{ij}^{\alpha\beta} = \hat{T}_{ji}^{\alpha\beta}$ should be taken into account.

Examining the three parallelization schemes discussed in the previous section, it can be seen that the first one (Fig. 1a), satisfies the first two conditions, but the matrix elements $\hat{T}_{ij}^{\alpha\beta}$ and $\hat{T}_{ji}^{\alpha\beta}$ should be computed by different processors. As a result, the total number of computations will be doubled. Some attempts have been made so that processors exchange instead of recompute matrix elements. However we found that the communication cost is high and increases significantly with the number of processors. The second parallelization scheme (Fig. 1b) does not satisfy the first and third conditions. As we show in the previous section, each processor in general will contain a different number of data blocks.

The third parallelization scheme appears to have some interesting features. According to the example in Fig. 1c, only the matrix elements $\hat{T}_{ij}^{\alpha\beta}$, with $i > j$ will be computed. Also every processor computes the same amount of matrix elements ($2 \times D/4 \times D/4$ in the example). Hence this algorithm meets all three requirements set above. After the construction of the matrix $\hat{T}_{ij}^{\alpha\beta}$ in this way, each processor has to perform four matrix-vector multiplications of size $(D/4 \times D/4) \times (D/4)$. The resulting $(D/4)$ vector for each multiplication is stored in the proper position of a vector of size D. At the end of the computations from all processors, the global sum of vectors will yield the final result. This procedure is illustrated in Fig. 2 and it has been implemented in a computer code. This algorithm appears to be superior to the previous two and it has been used in the present study.

After the end of the matrix–vector multiplication and the calculation of the remaining terms of Eq. 3 in the real and reciprocal space, a new electric field $E_{i,\alpha}$ will be obtained that will be used in the next iteration, until convergence of the induced dipole moments is reached.

So far we have discussed only the parallel implementation of the real terms appearing in the equations of Sect. 2. We can see, however, that the static and induced terms for both ϕ_i and $E_{i,\alpha}$ contain terms in reciprocal space that should be evaluated in each of the three steps that are required for calculation of the energy and forces. These calculations are easily parallelized as they involve a single summation over the k-vectors. As the number of k-vectors, n_k, is usually between a few hundred and a few thousand, this is significantly larger than the number of processors P. The sum over the k-vectors can be divided into P blocks and each processor calculates only a partial sum of the n_k/P terms.

In a more efficient approach, the total number of k-vectors can be divided in B_k blocks, where $B_k > P$. A global-shared counter is used by all processors to pick the block it would be working on in the same fashion it was described in the previous section. By doing this, the amount of work corresponding to each processor is not predetermined. The advantage of this method is that any load imbalance induced in the statically determined calculations performed in the real space, is absorbed by the calculations in the reciprocal space. For example, the processor to finish the calculations in the real space first, will calculate a larger part of the sum in the reciprocal space as opposed to the processor that finished the real space calculations last. This is shown schematically in Fig. 3. It should be noted that for an efficient implementation of the

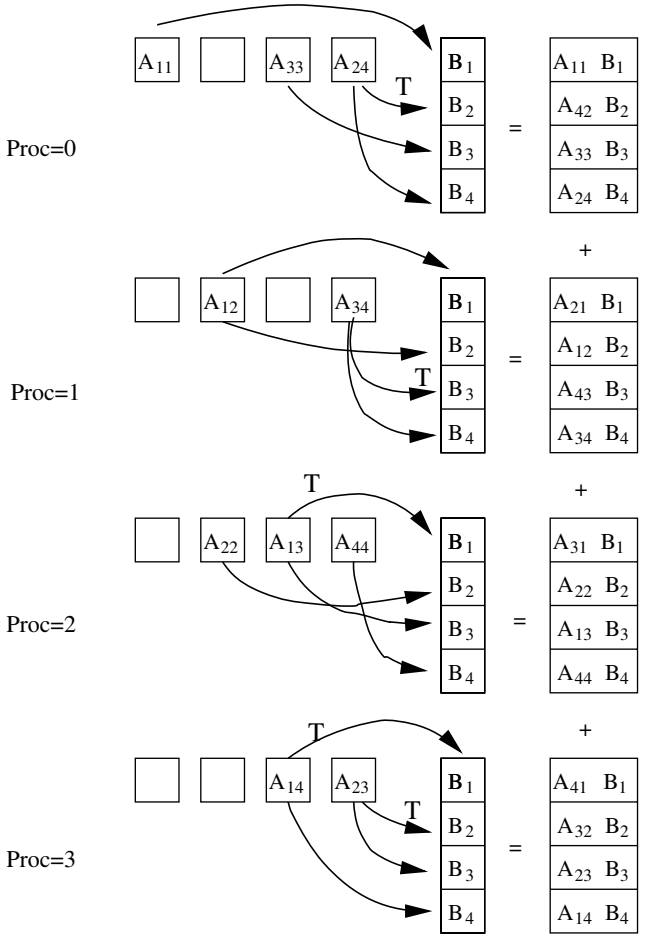

Fig. 2 An example of a matrix–vector multiplication using 4 processors. Each processor contains the data blocks shown in Fig. 1c. The *symbol (T)* denotes the multiplication of the transpose of a matrix with a vector. Note that indices i and j of matrix A (A_{ij}) are not referring to matrix elements, but to the divided blocks

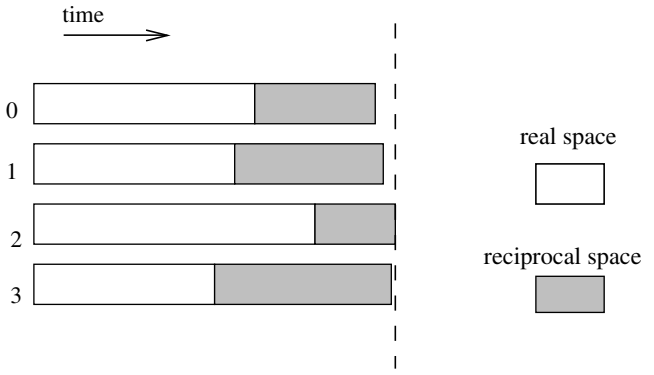

Fig. 3 Load balance for the possible imbalance introduced during the calculations in the real space is achieved during the calculations in the reciprocal space. The time required from each processor to complete the calculations in the real space is different. However, by properly adjusting the amount of work in the reciprocal space the total time required from all processor at the end of both these calculations is similar

sum in the reciprocal space, trigonometric formulas could be used for the evaluation of $e^{i\mathbf{k}\cdot\mathbf{r}_i}$ terms. A fortran code is given in Appendix 1 as an example. It should also be mentioned that there are several methods, such as the particle mesh Ewald method [22], which can be used for the calculations in the reciprocal space in a more efficient way. Future implementation of these methods does not require any further modifications of the presently proposed parallelization scheme.

5 Implementation and performance evaluation

5.1 Empirical potentials

In order to test the efficiency of our parallel algorithm we have used the Dang–Chang (DC) [23], and the version 2.1 of the Thole-type, flexible, (TTM2.1-F) [24] polarizable water models. The first has one polarizable site whereas the second has three polarizable sites per molecule. We have also employed the pairwise–additive flexible model TIP4F [25] as a point of reference. All these empirical potentials are 4-site models, with a fictitious massless site (M-site). For the DC and TTM2.1-F models the M-site is located in the bisector of the HOH angle. The position vector of the M-site is determined via the holonomic constraint of Reimers et al. [26]

$$\mathbf{r}_M = \mathbf{r}_O + \frac{\gamma}{2}(\mathbf{r}_{H_1O} + \mathbf{r}_{H_2O}). \tag{12}$$

The TTM2.1-F model uses the Partridge and Schwenke [27] potential energy surface for the description of the monomer. For the TIP4F potential the interactions between the sites (three atoms, M-site) within each monomer are described by harmonic oscillators. On the other hand, the DC treats the water molecule as a rigid body. However, as the goal of this work is to directly compare the efficiency of our algorithm during MD simulations for different systems at the same conditions, we have also treated DC as a flexible model [hereafter denoted as DC(F)], in which the intramolecular OH stretches and HOH bend are described by harmonic oscillators. Details related to the analytical functions that each model uses for the description of the intramolecular and van der Waals (vdw) interactions are not given here as from the technical point of view their implementation is similar.

The charge and polarizable sites for the three models are illustrated in Fig. 4, while the corresponding parameters are given in Table 1. Due to the different treatment of the electrostatic interactions in the TTM2.1-F model, which uses smearing instead of point charges, additional details are presented in Appendix 2.

In our tests we have considered three system sizes containing 128, 256 and 512 water molecules in a cubic simulation box. In all tests the density of the system was 0.997g/cm^3. Spherical cutoffs of $R_c = 9.0$ and 10.0 Å were chosen for the boxes containing 256 and 512 molecules respectively, while the box with 128 molecules included interactions between all molecules. The k-vectors in the reciprocal space were

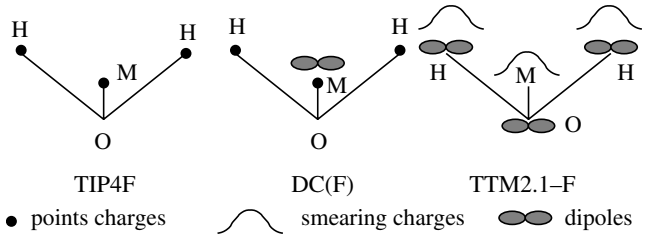

Fig. 4 The three interaction potentials used in this study. The charge and polarizable sites are shown

Table 1 Charges (q) and polarizabilities (a) for the water models

	TIP4F	DC(F)	TTM2.1-F
q_O	0.0	0.0	0.0
q_H	0.511	0.519	0.578
q_M	−1.022	−1.038	−1.156
a_O	0.0	0.0	0.837
a_H	0.0	0.0	0.496
a_M	0.0	1.444	0.0

Note that the values of the charges of the TTM2.1-F potential depend on the water monomer configuration (see Appendix 2 for details). In the table they correspond to the minimum energy conformation

2,442, 3,215, and 4,608 for the systems with 128, 256 and 512 molecules, respectively. Using the Nosé thermostat [28], a constant volume and temperature (NVT) MD simulation of 1, 000 steps at $T = 298$ K was performed for all systems with a time-step of 0.2 fs. The velocity Verlet [29] algorithm was employed for the integration of the trajectories. For the polarizable models the required accuracy for the induced dipole moments was 10^{-6} Debye. We found that at these simulation conditions, 15–20 iterations are required for the convergence of the induced dipoles when using as initial guess of the total electric field $\mathbf{E}^{(0)} = \mathbf{E}^{\text{stat}}$. A better estimation of $\mathbf{E}^{(0)}$ has been proposed by Ahlström et al. [30] using a third-order prediction scheme during the MD trajectory. At time t, the $\mathbf{E}^{(0)}(t)$ is given by

$$\mathbf{E}^{(0)}(t) = \sum_{n=1}^{4} \lambda_n \mathbf{E}(t - n\delta t), \tag{13}$$

where δt is the time step and $\lambda_1 = 4$, $\lambda_2 = -6$, $\lambda_3 = 4$, $\lambda_4 = -1$. It was found that in this case the required number of iterations to achieve the same accuracy is reduced to 3–4. This method was used in all benchmarks. The timings reported in this paper also include the calculation of the virial [11].

5.2 Computer platforms and tools

The benchmarks were performed in two different computer platforms at Pacific Northwest National Laboratory (PNNL). The first one is a HP cluster in which each cluster node has dual IA64 Madison 1.5 GHz CPUs on a HP ZX1 Chipset (from now on denoted as HP/IA64). Each Madison processor has an individual level 1 (L1) and level 2 (L2) caches and a 6 MB, on die, level 3 (L3) cache. The nodes in the cluster are interconnected by the Quadrics QsNetII network. The interconnect connects to a system via a 133 MHz PCI-X bus. Each node has at least 6 GB of memory. The second platform (denoted as IBM/P5) is an IBM eserver p5-595. It is a 64-way SMP with 32 1.65 GHz 64-bit Power5 processors. Each POWER5 chip has 2 cores and each of these processing cores has an individual L1 cache, shared L2 cache and a shared L3 cache of 36 MB.

For the implementation of the algorithm the global array (GA) toolkit was used. The GA toolkit provides an efficient and portable shared memory programming interface for distributed memory computers. Each process in a MIMD parallel program can asynchronously access logical blocks of physically distributed dense multi-dimensional arrays, without the need for explicit cooperation by other processes. GA provides interfaces that allow distribution of data while maintaining the type of global index space and programming syntax similar to what is available when programming on a single processor. It also provides interfaces to operate on data atomically, which is needed for the "atomic" counter update mentioned in Section III. GA also provides interfaces for creating and destroying multi-dimensional arrays and several one-sided and collective operations on arrays and array sections. Compatibility of GA with MPI enables the programmer to take advantage of the existing MPI software/libraries when available and appropriate. The code was developed in fortran-90 using the Intel `efc` and the IBM `xlf` compilers for the HP/IA64 and IBM/P5 platforms respectively. Only the intrinsic fortran-90 mathematical functions were used.

5.3 Performance and scaling

The wall times of the benchmarks on the HP/IA64 and IBM/P5 platforms are shown in Tables 2, 3 and 4 for the three empirical potentials with three different system sizes.

Table 2 Wall-clock time (seconds) required for 1, 000 MD steps on the HP/IA64 and IBM/P5 system platforms as a function of the number of processors for a unit cell containing 128 water molecules

Model	Platform	1p	2p	4p	8p	16p	32p
TIP4F	HP/IA64	30.3	15.6	8.2	4.8	2.8	2.7
	IBM/P5	28.0	14.3	7.6	4.2	2.8	5.9
DC(F)	HP/IA64	103.7	50.0	26.1	14.6	8.4	7.2
		35.8	*15.4*	*8.0*	*4.6*	*2.8*	*2.2*
	IBM/P5	96.1	48.5	25.3	13.4	7.9	11.1
		25.1	*12.5*	*6.5*	*3.3*	*1.8*	*1.1*
TTM2.1-F	HP/IA64	310.0	133.7	55.0	29.0	16.2	12.6
		152.6	*54.8*	*16.1*	*8.7*	*4.8*	*3.2*
	IBM/P5	224.6	114.0	58.3	30.0	16.5	16.5
		54.1	*28.0*	*14.1*	*7.2*	*3.7*	*3.7*

The three blocks of the table refer to the TIP4F, DC(F), and TTM2.1-F potentials, respectively. The required time for the computation of the induced dipole moments with the iterative procedure is shown with italics

Table 3 Same as Table 2 for a unit cell of 256 water molecules

Model	Platform	1p	2p	4p	8p	16p	32p
TIP4F	HP/IA64	85.7	43.6	22.4	11.9	6.8	4.5
	IBM/P5	78.8	40.0	20.3	10.7	6.2	7.7
DC(F)	HP/IA64	265.2	126.7	61.2	32.0	17.7	11.1
		73.3	*29.6*	*11.8*	*6.3*	*3.7*	*2.5*
	IBM/P5	239.9	121.1	60.7	31.3	17.0	14.2
		40.5	*20.8*	*10.1*	*5.2*	*2.7*	*1.5*
TTM2.1-F	HP/IA64	839.6	426.1	188.9	77.8	39.0	26.6
		377.1	*184.3*	*67.9*	*18.1*	*8.7*	*6.8*
	IBM/P5	605.3	293.4	148.6	74.7	39.3	26.1
		128.8	*56.1*	*29.4*	*14.2*	*7.6*	*4.4*

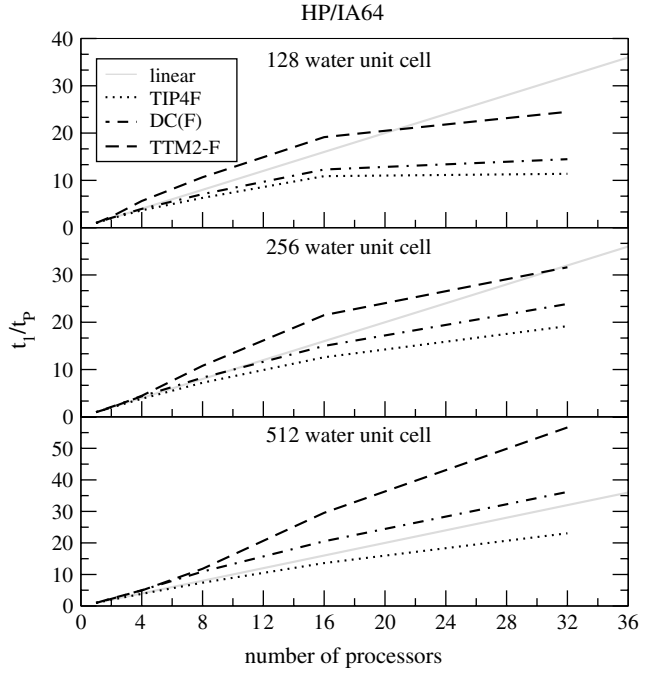

Fig. 5 Performance of the parallel code on the HP/IA64 platform as a function of the number of processors for the three different potentials and the three different simulation boxes. The *solid lines* corresponds to the ideal (linear) scaling, the *dotted lines* to the TIP4F, the *dot–dashed line* to DC(F), and the *dashed line* to the TTM2.1-F potential

The speed-up with respect to serial wall time t_1 is defined as

$$s = \frac{1}{P}\left(\frac{t_1}{t_P}\right), \tag{14}$$

where P is the number of processors and t_p is the corresponding execution time in parallel. For the HP/IA64 platform the relative timings with respect to serial execution (t_1/t_P) are shown in Fig. 5 and for the IBM/P5 on Fig. 6. It can be seen that in several cases the polarizable models show a super-linear speed-up.

By comparing the results of Tables 2, 3 and 4 we see that for the pairwise–additive potential, the times do not differ significantly for the HP/IA64 and IBM/P5 platforms. However, for the polarizable models the results are quite different. Usually for a small number of processors the IBM/P5 is much faster than the HP/IA64, while for a larger number of processors in some cases the HP/IA64 outperforms. For example, in the case of a cell with 512 molecules and the DC(F) empirical potential on 1 processor the IBM/P5 is faster than the HP/IA64 (790 sec and 1113 sec respectively), while for 32 processors the HP/IA64 is faster than the IBM/P5 (30.8 sec and 43.3 sec respectively). The total wall times for the simulation cell of 512 molecules are shown in Fig. 7.

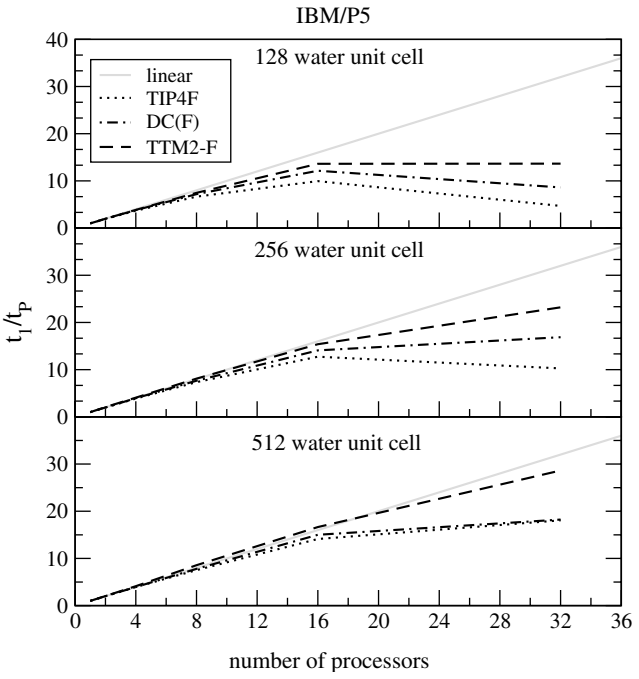

Fig. 6 Same with the Fig. 5 for the IBM/P5 platform

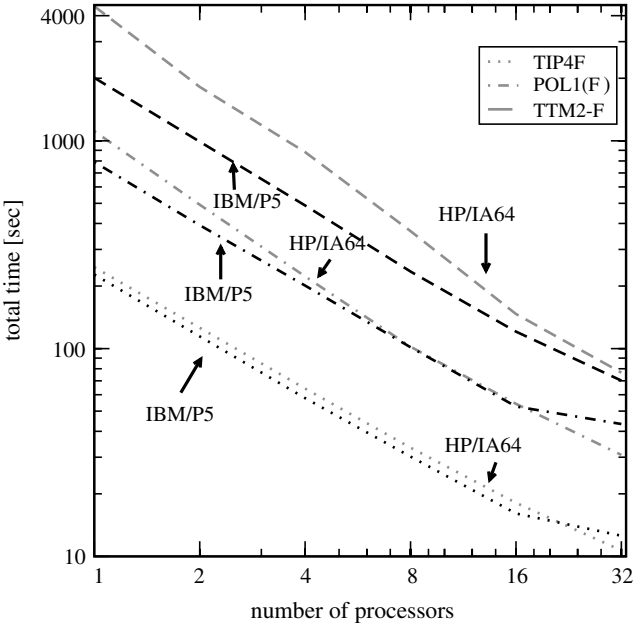

Fig. 7 Time to solution for 1, 000 MD steps in a system with 512 water molecules on the HP/IA64 and IBM/P5 platforms. The times are shown for the three empirical potentials on 1–32 processors (Table 4)

By examining the time needed for the calculation of the induced dipoles (shown in italics in the tables) we see that the IBM/P5 is faster. However, by taking the difference between total wall time and the time spent in the computation of the induced dipole moments, we see that it is quite similar for the two platforms. This can be also seen for the pairwise additive TIP4F potential of Fig. 7. Hence, it is only the performance of the matrix–vector multiplication that differs significantly between the two platforms. This is a result of the size of the cache memory of each processor, particularly the L3 cache as it will be discussed in more detail in the next section.

By comparing the wall times we can see that they are similar for the TIP4F on 1 processor, the DC(F) (one polarizable site) on 4 processors and the TTM2.1-F (three polarizable sites) on 8 processors. As a conclusion, 4 and 8 are roughly the number of processors required for the DC(F) and TTM2.1-F polarizable models in order to be able to perform MD simulations with speeds comparable to a serial execution of the pairwise–additive (TIP4F) model.

Table 4 Same as Table 2 for a unit cell of 512 water molecules

Model	Platform	1p	2p	4p	8p	16p	32p
TIP4F	HP/IA64	246.4	125.9	64.2	33.4	18.1	10.7
	IBM/P5	227.0	114.5	57.8	30.2	16.1	12.6
DC(F)	HP/IA64	1112.8	492.5	222.6	102.1	54.2	30.8
		552.9	*205.7*	*78.7*	*28.8*	*15.6*	*9.2*
	IBM/P5	790.5	392.4	201.0	101.5	52.6	43.3
		211.7	*102.9*	*55.2*	*27.4*	*13.8*	*13.4*
TTM2.1-F	HP/IA64	4339.0	1816.5	882.9	367.4	146.8	76.6
		2944.0	*1085.3*	*514.3*	*181.2*	*50.5*	*26.3*
	IBM/P5	2009.5	990.3	488.6	234.9	120.8	70.2
		650.0	*314.0*	*152.3*	*67.2*	*34.6*	*20.2*

6 Discussion and conclusions

The algorithm presented in this work was designed in order to minimize the communication between processors as much as possible, take advantage of the symmetries of the problem by avoiding the recalculation of pair energies, forces or matrix elements and balancing the workload of every processor by properly combining the computations in the real and the reciprocal space in a single step. The results presented in the previous section show that by taking advantage of the modern processor architectures it is possible to design algorithms for the efficient parallelization of polarizable empirical potentials that show super-linear scaling with the number of processors, as shown in Fig. 5. This result is an interesting feature of our algorithm and it will be further analyzed in terms of the architecture of the two computer platforms which were used in this study.

The size of the L3 cache for Power5 is about six times larger than that for the IA64. As the number of processors increases, the size of data being operated on becomes smaller. If the size of the data being used fits in the cache, the impact of the cache and its size will not affect the execution time of the induced dipole moments (shown in italics in the Tables 2, 3, 4). Hence as the number of processors increases and the problem size becomes smaller, both platforms produce much closer and relatively linear execution times. Even for 1 processor runs on the Power5, the data being operated on fits better in its large (36 MB) L3 cache. Hence the speed up seen on this platform for TIP4F, DC(F) and TTM2.1-F potentials with 128, 256 and 512 water molecules is linear. This is however not the case for the IA64 with its smaller (6 MB) L3 cache. In particular, it can be seen from the execution time of induced dipole moments for the 512 molecules supercell case (shown in italics in Table 4) that for the DC(F) and TTM2.1-F the wall-clock time does not scale linearly for smaller processor count (visible for up to 4 processors). This is the effect of L3 cache misses and the fact that data being operated on was much larger for the cache and resulted in several L3 cache misses. To validate our assumption we profiled the number of L3 cache misses on this system for the 512 water molecule supercell case with the TTM2.1-F potential. Figure 8 shows the number of L3 cache misses when using the TTM2.1-F potential for a supercell of 512 water molecules on 1, 2 and 4 processors. It can be seen that the number of L3 cache misses when running on 2 processors versus 1 processor goes down by a factor of 2 on Power5 architecture while on the IA64 it drops by a factor of 15, explaining the super-linear behavior for this case. Similarly, when executing on 4 processors versus 2 processors, the number of L3 cache misses drops by factors of 2 and 4 for the Power5 and IA64 architectures, respectively.

It should be emphasized that all the benchmarks were performed with flexible water models that require a small integration time step, δt, during a MD simulation. In the case of rigid models it is possible to increase the time step by about one order of magnitude to $1-2$ fs, in this way decreasing the number of steps required for a simulation of a specific length. However, in this case the number of iterations required for the convergence of the induced dipoles will increase significantly. If we treat the water monomers of the previous polarizable models as rigid bodies, the number of iterations required increased to 10–12 instead of 3–4 with flexible water monomers and 0.2 fs time step. In this case the efficiency of the proposed algorithm will be much more advantageous when it is executed across several processors.

The efficiency of the algorithm proposed here is based on the fact that the dipole tensor matrix elements are stored in an array and are not recomputed during the iterative procedure required for the calculation of the induced dipoles. In this way the total time-to-solution is reduced. Additional gain comes from the efficiency that modern processors can execute the matrix-vector multiplication, which in several cases results to a superlinear speed-up with increasing number of processors. However, this scheme has intrinsic limitations when the size of the physical system becomes significantly larger than the typical sizes used in this study: (i) the memory requirements increase as $\approx N^2$ limiting in this way the maximum size of the system. (ii) the dipole-tensor array becomes sparse. The amount of the non-zero matrix elements depends on the dimension of the box L, as well as the radius of the

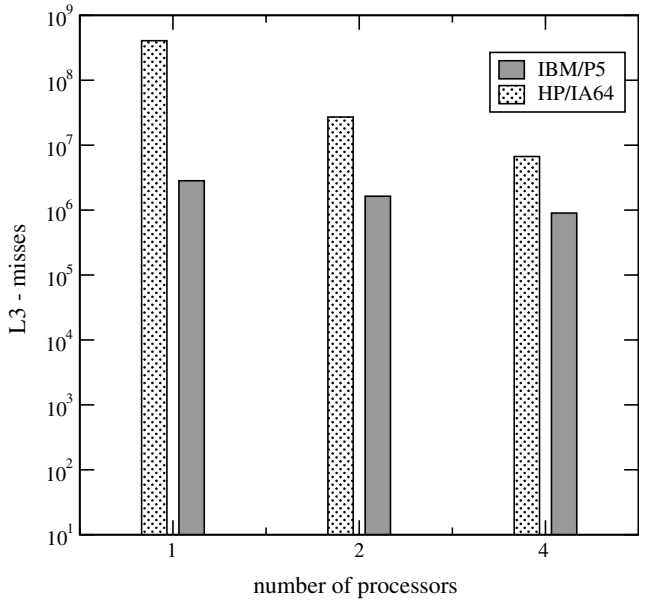

Fig. 8 Number of the L3 cache misses on the two computer platforms for 1, 2 and 4 processors during the calculation of the induced dipoles for a simulation box of 512 water molecules and the TTM2.1-F potential

sphere R_c that the interactions in the real space are considered for every atom. The ratio of the number of interactions considered over the total number of interactions for all atoms in the box is given by: $a = (4\pi/3)(R_c/L)^3$ (where for $a = 1$ all interactions are considered). For a system for which $R_c = L/2$ about half of the matrix elements are non-zero ($a = 0.52$). When the Ewald summation technique is employed, typical values used for R_c are between 9.0 and 12.0Å. If $L/2$ is significantly larger than R_c then most of the matrix elements of the dipole tensor will be zero. In this case, it is possible that the multiplication of the non-zero matrix elements only (instead of the total matrix) with the dipole moment vector will become a more efficient approach.

Based on the guidelines of the current implementation it would possible for these system sizes to develop parallel codes that require even less time-to-solution. The improvements proposed here are the particle-mess Ewald method (pme) [22] for the treatment of the long-range corrections and the link-cell method [31] in combination with the Verlet list. These improvements however are expected to become important for systems of much larger sizes than the ones considered in this study. Despite the limitations of the algorithm discussed earlier for larger systems, we believe that it has a wide range of applicability for many physical systems of small to medium size (i.e. up to a few thousand polarizable sites).

Acknowledgements We are grateful to G. Schenter and S. Elbert for stimulating discussions and E. Aprà for the contribution in the early development of the parallel codes. This work was supported by the Division of Chemical Sciences, Geosciences and Biosciences, Office of Basic Energy Sciences, US Department of Energy. Battelle operates the Pacific Northwest National Laboratory for the Department of Energy. This research was performed in part using the Molecular Science Computing Facility (MSCF) in the William R. Wiley Environmental Molecular Sciences Laboratory, a national scientific user facility sponsored by the U.S. Department of Energy's Office of Biological and Environmental Research and located at the Pacific Northwest National Laboratory.

Appendix 1

Here we outline the approach for performing parallel calculations in the reciprocal space using fortran code. The symbol "!" denotes a comment. We note that $e^{ix} = \cos(x) + i\ \sin(x)$.

In the first step, the number of k-vectors (`nvec`) and their indices will be determined and stored in `kv`. Usually they do not change during a MD simulation and the following subroutine is executed only at the beginning of the code from all processors.

```
nvec=0
do nx=0, max_nx
 do ny=-max_ny, max_ny
  nxy=nx**2 + ny**2
  do nz=-max_nz, max_nz
    if (nxy + nz**2 < max_n2)
     nvec=nvec+1
     kv(1:3, nvec) = (/nx, ny, nz/)
   endif
  enddo ! nz
 enddo ! ny
enddo ! nx
```

At this point each processor pre-calculates and stores some trigonometric terms. In this way it can take advantage of trigonometric formulas for the fast calculation of $e^{i\mathbf{k}\cdot\mathbf{r}}$ during the main calculation. The following code calculates the $\cos(k_x x)$ and $\sin(k_x x)$ (denoted as `cx` and `sx` in the code) for every atom i. A similar code should be used for the calculation of `cy`, `sy`, `cz` and `sz`. `Lx` is the size of the box in the x-direction, `Rx(i)` is the x-coordinate of each atom and `Natoms` is the total number of atoms.

```
do i=1, Natoms
 cx(i,0)=1.0
 sx(i,0)=0.0
 sx(i,1)=sin(2.0*PI*Rx(i)/Lx)
 cx(i,1)=cos(2.0*PI*Rx(i)/Lx)
enddo
do n=2, max_nx
 do i=1,Natoms
  cx(i,n)=cx(i,n-1)*cx(i,1)-sx(i,n-1)
  *sx(i,1)
  sx(i,n)=cx(i,n-1)*sx(i,1)+sx(i,n-1)
  *cx(i,1)
 enddo
enddo
```

Finally the most time consuming part of the computation is the one used for the calculation of the reciprocal terms of ϕ_i, $E_{i,\alpha}$ and $E_{i,\alpha\beta}$. The total number of k-vectors is divided in blocks the number of which is determined by the user (`Nblocks`). At the beginning the "atomic" counter (`next`)

is initialized to one, by calling the GA subroutine: `ga_fill`. By reading and increasing by one the value of this counter (using the `nga_read_inc`) the block (`iblock`) that each processor will work on is determined. The block contains the k-vectors with indices between `ivec1` and `ivec2`. The $\cos(\mathbf{k}\cdot\mathbf{r})$ and $\sin(\mathbf{k}\cdot\mathbf{r})$ (`cr` and `sr` respectively) are calculated for this block after an intermediate calculation of $\cos(k_x x+k_y y)$ and $\sin(k_x x+k_y y)$, (denoted as `cxy` and `sxy`, respectively). By examining if the last two quantities have been calculated in the previous step (the variables `lastnx` and `lastny` are used for this reason) significant amount of computations can be avoided.

```
call ga_fill(next, 1, 1)
blocksize = nvec / Nblocks
lastnx = -10000
lastny = -10000

iblock = nga_read_inc(next, 1, 1)

do while (iblock<=Nblocks)
 ivec1 = (iblock-1)*blocksize+1
 ivec2 = min(ivec1+blocksize, nvec)
 do n=ivec1, ivec2
  nx = kv(1,n)
  ny = kv(2,n)
  nz = kv(3,n)

 if (nx/=lastnx .or. ny/=lastny) then
  do i=1, Natoms
  cxy(i)=cx(i,nx)*cy(i,ny)-sx(i,nx)
  *sy(i,ny)
  sxy(i)=sx(i,nx)*cy(i,ny)+cx(i,nx)
  *sy(i,ny)
 enddo
endif
 do i=1,Natoms
  cr(i)=cxy(i)*cz(i,nz)-sxy(i)*sz(i,nz)
  sr(i)=sxy(i)*cz(i,nz)+cxy(i)*sz(i,nz)
 enddo
 lastnx=nx
 lastny=ny
!
! A CALCULATION IN THE RECIPROCAL SPACE
! CAN BE PERFORMED AT THIS POINT
!
 iblock=nga_read_inc(next, 1, 1)
 enddo ! do n=ivec1, ivec2
enddo ! do while (block<=Nblocks)
```

The previous code can be used for every calculation in the reciprocal space and differs only in the quantity that will be computed. As an example, the following code can be used for the calculation of Q^q in Eq. (7) (real and imaginary parts):

```
do i=1, Natoms
   real_Qq = real_Qq + charge(i)*cr(i)
  imag_Qq = imag_Qq + charge(i)*sr(i)
enddo
```

Appendix 2

In this section we briefly describe the details of the TTM2.1-F potential of Fanourgakis and Xantheas [24] which is a recent revision of earlier work by Burnham et al. [32] and Burnham and Xantheas [33] related to the treatment of the electrostatic interactions.

In the case of point charges and dipoles the interaction tensor $\hat{T}_{ij}$ is given by:

$$\hat{T}_{ij} = \frac{1}{r_{ij}}. \tag{15}$$

For the Ewald summation, the modified interaction tensor given by

$$\hat{T}_{ij} = \frac{\mathrm{erfc}(\kappa r_{ij})}{r_{ij}}, \tag{16}$$

should be introduced, where κ is the Ewald parameter and r_{ij} is the distance between atoms i and j. The $\hat{T}^{\alpha}_{ij}$, $\hat{T}^{\alpha\beta}_{ij}$ and $\hat{T}^{\alpha\beta\gamma}_{ij}$ can been easily calculated using the recursive formula of Smith given in Ref. [34].

In the case of the TTM2.1-F model, smearing instead of point charges have been used according to the Thole's model [35]. Among the different formulas proposed by Thole, TTM2.1-F adopts the following charge distribution:

$$\rho = \frac{1}{A^3}\frac{3a_s}{4\pi}\exp\left(-a_s u^3\right), \tag{17}$$

where $u = r_{ij}/A$, $A = (\alpha_i\alpha_j)^{1/6}$, α_i, α_j are the polarizabilities of atoms i and j, respectively, and a_s is a dimensionless parameter that determines the width of the distribution. In this case the interaction tensor $\hat{T}_{ij}$ is given by:

$$\hat{T}_{ij} = \frac{1-\exp(-a_s u^3) + a_s^{1/3} u\Gamma(2/3, a_s u^3)}{r_{ij}}, \tag{18}$$

where $\Gamma(a, x)$ is the "incomplete Gamma function". A careful analysis of the previous equation shows that only the first term (r_{ij}^{-1}) is important for large separations of the atoms. This significantly simplifies the modified tensor $\hat{T}_{ij}$ that should be used in the Ewald summation:

$$\hat{T}_{ij} = \frac{\mathrm{erfc}(\kappa r_{ij}) - \exp(-a_s u^3) + a_s^{1/3} u\Gamma(2/3, a_s u^3)}{r_{ij}}. \tag{19}$$

In our implementation we have considered that only the first term of Eq. (19) survives when the distance r_{ij} is larger than 5.0 Å. It should be mentioned also that in the TTM2.1-F potential, different smearing parameters have been used for the charge–charge, charge–dipole ($a_s = 0.2$), and dipole–dipole ($a_s = 0.3$) interactions.

As it was mentioned before the TTM2.1-F uses the model of Partridge and Schwenke [27] for the description of the intramolecular interactions. This model has been developed in order to reproduce high level ab-initio energies and dipole moments. In contrast to the other empirical potentials that are assigning constant fractional charges to each atom, this

model is able to reproduce the dipole moment surface of water monomer by assigning to each atom fractional charges that depend on the geometry of the molecule. This introduces a small complication in the calculation of the forces as the gradients of the charges with respect to the coordinates of the atoms $\mathbf{r}_i$, are non-zero. Details and an extensive discussion can be found in previous works [24].

References

1. MPI (1995) A message-passing interface standard, MPI forum
2. Nieplocha J, Harrison RJ, Littlefield RJ (1994) In: Proceedings of Supercomputing, p. 340
3. Nieplocha J, Harrison RJ, Littlefield RJ (1996) J Supercomput, 10:169
4. Nieplocha J, Tipparaju V, Krishnan M, Trease, H, Aprà, E (2006) Int J High Perform Comput Appl 20:203
5. Brooks BR, Bruccoleri RE, Olafson BD, States DJ, Swaminathan S, Karplus M (1983) J Comp Chem 4:197
6. Pearlman DA, Case DA, Caldwell JW, Ross WS, Cheatham III TE, DeBolt S, Ferguson D, Seibel, G, Kollman P (1995) Comp Phys Commun 91:1
7. Grossman JC, Schwegler E, Draeger EW, Gygi F, Galli G (2004) J Chem Phys 120:300
8. Allesch M, Schwegler E, Gygi F, Galli G (2004) J Chem Phys 120:5192
9. Barker JA, Watts RO (1969) Chem Phys Lett 3:144
10. Rahman A, Stillinger FH (1971) J Chem Phys 55:3336
11. Nymand TM, Linse P (2000) J Chem Phys 112:6152
12. Plimpton S (1995) J Comp Phys 117:1
13. Snir M (2004) Theory Comput Systems 37:295
14. Heffelfinger GS (2000) Comp Phys Comm 128:219
15. Shaw DE (2005) J Comp Chem 26:1318
16. Verlet L (1969) Phys Rev 159:98
17. Allen MP, Tildesley DJ (1987) Computer simulation of liquids. Oxford University Press, New York
18. Willebeek-LeMair MH, Reeves AP (1993) IEEE Trans Parallel Distrib Syst 4:979
19. Harrison RJ, Guest MF, Kendall RA, Bernholdt DE, Wong AT, Stave M, Anchell JL, Hess AC, Littlefield RJ, Fann GL, Nieplocha J, Thomas GS, Elwood D, Tilson JL, Shepard RL, Wagner AF, Foster IT, Lusk, E. and Stevens R, J Comp Chem 17:124
20. Press WH, Flannery BP, Teukolsky SA, Vetterling WT (1986) Numerical Recipes. Campridge University Press, New York.
21. Wadleigh KR, Crawford IL (2000) Software optimization for high performance computing. Prentice Hall Englwood Cliffs
22. Essmann U, Perera L, Berkowitz ML, Darden T, Lee H, Pedersen LG (1995) J Chem Phys 103:8577
23. Dang LX, Chang TM (1997) J Chem Phys 106:8149
24. Fanourgakis GS, Xantheas SS (2006) J Phys Chem A 110:4100
25. Mahoney MW, Jorgensen WL (2001) J Chem Phys 115:10758
26. Reimers JR, Watts RO, Klein ML (1982) Chem Phys 64:95
27. Partridge H, Schwenke DW (1997) J Chem Phys 106:4618
28. Nosé, S (1984) Mol Phys, 52:255
29. Swope WC, Andersen HC, Berens PH, Wilson KR (1982) J Chem Phys 76:637
30. Ahlström P, Wallqvist A, Engström S, Jönsson B (1989) Mol Phys 68:563
31. Quentrec B, Brot C (1973) J Comput Phys 13:430
32. Burnham CJ, Li J, Xantheas SS, Leslie M (1999) J Chem Phys 110:4566
33. Burnham CJ, Xantheas SS (2002) J Chem Phys 116:5115
34. Smith W (1982) CCP5 Info. Q 4:13
35. Thole BT (1981) Chem Phys 59:341